CPSC 213 – Assignment 3
Static Variables and C Pointers

Due: Wednesday, Oct 12, 2016 at 11:59pm
No late assignments accepted.

Overview

This assignmenth has three parts. First you will examine two C programs that access arrays and then translate them into assembly language. Then you will answer some questions about C pointer arithmetic. Finally, you will investigate dynamic arrays and pointers in C and then translate another portion of a C program into assembly language.

Part 1: Translating C Code into SM213 Assembly *[40%]*

The code file for this week is found in *a3_code.zip*.

This file contains a directory named `examples` that contains a set of example C programs and their sm213 assemble-code implementation. The code file also contains two C files named `a2_1.c` and `a2_2.c`.

Carefully examine the example programs. Run the assembly versions in the simulator, step by step. Compare their execution to the C versions. Get comfortable with this translation from C to assembly and with how assembly executes in the machine.

Now, you're that you're getting comfortable, take the first of the C files that don't have assembly, `a2_1.c` and produce your own sm213 file that does what this program does. Every line of your assembly file must have a comment. The comment should explain what the line does by referencing C syntax whenever possible. For example, if an assembly instruction loads the value of a variable into a register, the comment should name that variable. Do the same for the other C file. The lines get progressively harder so do them in order.

Be sure to end your programs with a "halt" instruction so that the simulated CPU stops when it reaches the end of your program. If you don't do this, then it will keep running, interpreting whatever if finds in memory as instructions, probably doing very strange things.

Note that these snippets are not complete programs. Imagine that they were extracted from a larger program that did things like initialize the value of global variables etc. You should assume that pointer variables have been initialized to hold the address of an array of adequate size. That means that you need to create the array statically in the assembly and set the pointers to the static address of the array. This isn't what a real compiler would do, but by doing this you will be able to examine all of the data the program accesses in the simulator. The simulator only shows memory address that are specifically named in the assembly file.

Part 2 *[15%]*

Answer These Questions

Place your answers in files named `Q1.txt`., `Q2.txt`, and `Q3.txt`. Note that if you aren't sure of the answers, you can always write C program and have it tell you the answer.

Assume that the array `a` is declared and initialized like this:

```
int a[10] = {0,1,2,3,4,5,6,7,8,9};
```

1. What is the value of: `*(a+3)`?
2. What is the value of: `&a[7] - &a[2]`?
3. What is the value of: `*(a + (&a[7]-a+2))`?

Part 3 *[45%]*

Overview

In *a3_code.zip*, you will find a file named `bubble_sort_static.c` that takes as input up to 4 integers on the command line and uses the *Bubble Sort* algorithm to sort them. This is your first C program, here are a few of the basics you'll need to get started.

Compiling and running C programs is done on the command line. You can edit your program with any editor you like, but be sure that it produces ***plain-text*** output, not the sort of thing that a word processor produces.

You compile this program like this

```
gcc -std=gnu11 -o bubble_sort_static bubble_sort_static.c
```

You run it like this:

```
./bubble_sort_static 78 3 43 1
```

Read this C file carefully. You will see that the `main` function has two parameters: `argc` and `argv`. In Java it is similar: `main` has one parameter, an array of strings. That's what `argc` and `argv` are: `argc` is the number of stings found on the command line that caused the program to run (5 in the case of the example) and `argv` is a dynamic array that contains each of these strings. In C, a string is simply an array of characters that is terminated by the first `null` (i.e.,

0) in the string. Remember that C arrays do not have length, so this is sort of a substitute for that … for strings.

You will also see that `main` copies these values in a static global array named `val`. This array is statically assigned a length of 4 and so the input is limited to 4 numbers.

Finally, once `main` has copied the input values into `val`, it calls `sort` to sort the values and then prints out the result.

Step 1: Modify the C program to use a dynamic array *[15%]*

The first step is to create a new C program named `bubble_sort_dynamic.c` that removes the static-array limitation of the original version. The only change you need to make is to turn `val` into a dynamic array and to allow the program to sort arbitrary lists of integers provided on the command line (e.g., more than 4).

Recall that you need to allocate dynamic arrays dynamically by calling the procedure `malloc`. For example to allocate an array of 10 shorts you say:

```
      short s* = malloc (10 * 2);
```

Be sure to test your program.

Step 2: C Pointers *[15%]*

Now, create another C program called `bubble_sort_awesome.c` that extends your earlier work to remove all square brackets from your program. All access to the arrays must be made using pointer arithmetic and the dereference operator (i.e., "`*`"). Test this too.

Step 3: Now for Assembly *[15%]*

Now, we'd like to ask you to implement the `sort` algorithm in assembly, but we haven't talked about loops or if statements yet, so lets take one (okay two) steps back from that. Instead, implement the swap portion of the inner loop in sm213 assembly code.

```
      t      = val[i];
      val[i] = val[j];
      val[j] = t;
```

Use the following data declarations (you may need more than this):

```
      i:   .long 0
      j:   .long 0
      val: .long 0 // …
```

Implement this two ways. Test your code with various values of i, j and val. Place your two solutions in files named `bubble_sort_static.s` and `bubble_sort_dyanmic.s`.

What to Hand In

Use the `handin` program; the command-line version is strongly preferred.

The assignment directory is `~/cs213/a3`, it should contain the following ***plain-text*** files.

1. `README.txt` that contains the name and student number of you and your partner
2. `PARTNER.txt` containing your partner's CS login id and nothing else (i.e., the 4- or 5-digit id in the form a0z1). Your partner should not submit anything.
3. `a2_1.s` and `a2_2.s` from Part 1. You must submit these as part of Assignment 3 even if you submitted versions of these files for Assignment 2.
4. `Q1.txt`., `Q2.txt`., and `Q3.txt` with your answers to Part 2.
5. `bubble_sort_dynamic.c` and `bubble_sort_awesome.c`.
6. `bubble_sort_static.s` and `bubble_sort_dynamic.s`.